Efficient coding of information: Huffman coding

Sridhara, Deepak

Abstract: In his classic paper of 1948, Claude Shannon considered the problem of efficiently describing a source that outputs a sequence of symbols, each associated with a probability of occurrence, and provided the theoretical limits of achievable performance. In 1951, David Huffman presented a technique that attains this performance. This article is a brief overview of some of their results

DOI: https://doi.org/10.1007/bf02837275

Efficient Coding of Information: Huffman Coding

Deepak Sridhara

In his classic paper of 1948, Claude Shannon considered the problem of efficiently describing a source that outputs a sequence of symbols, each associated with a probability of occurrence, and provided the theoretical limits of achievable performance. In 1951, David Huffman presented a technique that attains this performance. This article is a brief overview of some of their results.

Deepak Sridhara worked as a research associate in the Deparment of Mathematics at IISc between January and December 2004. Since January 2005, he has been a post-doctoral research associate in the Institute for Mathematics at Unviersity of Zurich. His research interests include coding theory, codes over graphs and iterative techniques, and information theory.

1. Introduction

Shannon's landmark paper 'A Mathematical Theory of Communication' [1] laid the foundation for communication and information theory as they are perceived today. In [1], Shannon considers two particular problems: one of efficiently describing a source that outputs a sequence of symbols each occurring with some probability of occurrence, and the other of adding redundancy to a stream of equally-likely symbols so as to recover the original stream in the event of errors. The former is known as the *source-coding* or data-compression problem and the latter is known as the *channel-coding* or error-control-coding problem. Shannon provided the theoretical limits of coding achievable in both situations, and in fact, he was also the first to quantify what is meant by the "information" content of a source as described here.

This article presents a brief overview of the source-coding problem and two well-known algorithms that were discovered subsequently after [1] that come close to solving this problem. Of particular note is Huffman's algorithm that turns out to be an optimal method to represent a source under certain conditions. To state the problem formally: Suppose a source U outputs symbols

Keywords
Entropy, source coding, Huffman coding.

[1] By a D-ary string we mean a sequence $(y_1, y_2, \ldots, y_n)$ where each y_i belongs to the set $\{0,1,2,\ldots,D-1\}$.

problem formally: Suppose a source U outputs symbols $u_1, u_2, \ldots, u_K$, belonging to some discrete alphabet $\mathcal{U}$, with probabilities $p(u_1), p(u_2), \ldots, p(u_K)$, respectively. The source-coding problem is one of finding a mapping from $\mathcal{U}$ to a sequence of D-ary strings[1], called codewords, such that this mapping is invertible. For the coding to be efficient, the aim is to find an invertible mapping (or, a code) that has the smallest possible average codeword-length (or, code-length). By the way this problem is stated, a source U can also be treated as a discrete random variable U that takes values from $\mathcal{U} = \{u_1, u_2, \ldots, u_K\}$ and that has a probability distribution given by $p(U)$, where $p(U = u_i) = p(u_i)$, for $i = 1, 2, \ldots, K$.

As an example, suppose a random variable U takes on four possible values u_1, u_2, u_3, u_4 with probability of occurrences $p(u_1) = \frac{1}{2}, p(u_2) = \frac{1}{4}, p(u_3) = \frac{1}{8}, p(u_4) = \frac{1}{8}$. A naive solution for representing the symbols $u_1, \ldots, u_4$ in terms of binary digits is to represent each symbol using two bits: $u_1 = 00, u_2 = 01, u_3 = 10, u_4 = 11$. The average code-length in this representation is two bits. However, this representation turns out to be a sub-optimal one. An efficient representation uses fewer bits to represent symbols occurring with a higher probability and more bits to represent symbols occurring with a lower probability. An optimal representation for this example is $u_1 = 0, u_2 = 10, u_3 = 110, u_4 = 111$. The average code-length in this representation is $1 \cdot \frac{1}{2} + 2 \cdot \frac{1}{4} + 3 \cdot \frac{1}{8} + 3 \cdot \frac{1}{8} = 1.75$ bits. Intuitively, the optimal number of bits needed to represent a source symbol u_i with probability of occurrence p_i is around $\log_2 \frac{1}{p_i}$ bits. Further, since u_i occurs with probability p_i, the effective code-length of u_i in such a code is $p_i \log_2 \frac{1}{p_i}$. Shannon's idea was to quantify the 'information' content of a source or random variable U in precisely these terms. Shannon defined the *entropy* of a source U to measure its information content, or equivalently, the entropy of

Intuitively, the optimal number of bits needed to represent a source symbol u_i with probability of occurrence p_i is around $\log_2 1/p_i$ bits.

a random variable U to measure its 'uncertainty', as $H(U) = \sum_i p_i \log_2 \frac{1}{p_i}$. The coding techniques discussed in this article are evaluated using this measure of 'information' content.

Shannon defined the *entropy* of a source U to measure its information content, or equivalently, the entropy of a random variable U to measure its 'uncertainty', as $H(U)=\Sigma_i p_i \log_2 1/p_i$.

This article is organized as follows. Section 2 introduces some definitions and notation that will be used in this paper. Section 3 provides the notion of typical sequences that occur when a sequence of independent and identically distributed random variables are considered. This section also states the properties of typical sequences which are used to prove the *achievability* part of Shannon's source-coding theorem, in Section 4. Section 5 introduces the notion of D-ary trees to illustrate the encoding of a source into sequences of D-ary alphabets. Another version of the source-coding theorem for the special case of prefix-free codes is presented here. A coding technique, credited to Shannon and Fano, which is a natural outcome of the proof of the source-coding theorem stated in Section 5, is also presented. Section 6 presents Huffman's optimal coding algorithm that achieves a lower average code-length than the Shannon-Fano algorithm. The paper is concluded in Section 7.

2. Preliminaries

Some basic definitions and notation that will be used in this article are introduced here.

Let X be a discrete random variable taking values in a finite alphabet $\mathcal{X}$. For each $x \in \mathcal{X}$ let $p(x)$ denote the probability that the random variable X takes the value x written as $Pr(X = x)$.

Definition 2.1. The expectation of a function $f(X)$ defined over the random variable X is defined by

$$E[f(X)] = \sum_{x \in \mathcal{X}} f(x)p(x)$$

Definition 2.2. The entropy $H(X)$ of a discrete random

The entropy $H(X)$ of a discrete random variable X is defined by $H(X) = -\sum_{x \in \mathcal{X}} p(x) \log p(x)$.

variable X is defined by

$$H(X) = -\sum_{x \in \mathcal{X}} p(x) \log p(x).$$

To avoid ambiguity, the term $p(x) \log p(x)$ is defined to be 0 when $p(x) = 0$. By the previous definition, $H(X) = -E[\log p(X)]$. For most of the discussion, we will use logarithms to the base 2, and therefore, the entropy is measured in bits. (However, when we use logarithms to the base D for some $D > 2$, the corresponding entropy of X is denoted as $H_D(X)$ and is measured in terms of D-ary symbols.) Note that in the above expression, the summation is over all possible values x assumed by the random variable X, and $p(x)$ denotes the corresponding probability of $X = x$.

We will use uppercase letters to indicate random variables, lowercase letters to indicate the actual values assumed by the random variables, and script-letters (e.g., $\mathcal{X}$) to indicate alphabet sets. The probability distribution of a random variable X will be denoted by $p(X)$ or simply p.

Extending the above definition to two or more random variables, we have

Definition 2.3. The joint entropy of a vector of random variables $(X_1, X_2, \ldots, X_n)$ is defined by

$$H(X_1, X_2, \ldots, X_n) =$$

$$- \sum_{(x_1, x_2, \ldots, x_n) \in \mathcal{X}_1 \times \mathcal{X}_2 \times \ldots \times \mathcal{X}_n} p(x_1, \ldots, x_n) \log p(x_1, x_n \ldots, x_n)$$

where $p(x_1, x_2, \ldots x_n) = Pr(X_1 = x_1, X_2 = x_2, \ldots X_n = x_n)$.

Definition 2.4. (a) A collection of random variables $\{X_1, X_2, \ldots X_n\}$ with X_i taking values from an alphabet $\mathcal{X}$ is called independent if for any choice $\{x_1, x_2, \ldots x_n\}$

with $X_i = x_i, i = 1, 2, \ldots n$, the probability of the joint event $\{X_1 = x_1, \ldots, X_n = x_n\}$ equals the product $\prod_{i=1}^{n} Pr(X_i = x_i)$.

(b) A collection of random variables $\{X_1, X_2, \ldots X_n\}$ is said to be identically distributed if $\mathcal{X}_i = \mathcal{X}$: that is, all the alphabets are the same, and the probability $Pr(X_i = x)$ is the same for all i.

A collection of random variables $\{X_1, X_2, \ldots, X_n\}$ is said to be identically distributed if $\mathcal{X}_i = \mathcal{X}$: that is, all the alphabets are the same, and the probability $Pr(X_i = x)$ is the same for all i.

(c) A collection of random variables $\{X_1, X_2, \ldots X_n\}$ is said to be independent and identically distributed (i.i.d.) if it is both independent and identically distributed.

An important consequence of independence is that if $\{X_1, X_2, \ldots X_n\}$ are independent random variables, each taking on only finitely many values, then for any function $f_i(.)$, $i = 1, 2, \ldots n$,

$$E[\prod_{i=1}^{n} f_i(X_i)] = \prod_{i=1}^{n} E[f_i(X_i)].$$

Thus if $\{X_1, X_2, \ldots X_n\}$ are independent and take values over finite alphabets, then the entropy

$$H(X_1, X_2, \ldots X_n) \equiv -E[\log p(X_1, X_2, \ldots X_n)] =$$

$$\sum_{i=1}^{n} E[\log p(X_i)] = \sum_{i=1}^{n} H(X_i).$$

If, in addition, they are identically distributed, then

$$H(X_1, X_2, \ldots X_n) = nH(X_i) = nH(X_1).$$

Note that if $X_1, \ldots, X_n$ are a sequence of i.i.d. random variables as above, then their joint entropy is $H(X_1, X_2, \ldots, X_n) = nH(X)$.

Definition 2.5. A sequence $\{Y_n\}$ of random variables is said to converge to a random variable Y *in probability* if for each $\epsilon > 0$

$$Pr(|Y_n - Y| > \epsilon) \longrightarrow 0, \text{ as } n \longrightarrow \infty,$$

The assertion that $\bar{X}_n \to E[X_1]$ with probability one is called the *strong law of large numbers* and the assertion that $\bar{X}_n \to E[X_1]$ in probability is called the *weak law of large numbers*.

and *with probability one* if

$$Pr(\lim_{n\to\infty} Y_n = Y) = 1.$$

It can be shown that if Y_n converges to Y with probability one, then Y_n converges to Y in probability, but the converse is not true in general.

The *law of large numbers* (LLN) of probability theory says that the *sample mean* $\bar{X}_n \equiv 1/n \sum_{i=1}^{n} X_i$ of n i.i.d. random variables $\{X_1, X_2, \ldots X_n\}$ gets close to $m = E[X_i]$ as n gets very large. More precisely, it says that $\bar{X}_n$ converges to $E[X_1]$ with probability 1 and hence in probability. Here we are considering only the case when the random variables take on finitely many values. The LLN is valid for random variables taking infinitely many values under some appropriate hypothesis (see R L Karandikar [4]). The assertion that $\bar{X}_n \longrightarrow E[X_1]$ with probability one is called the *strong law of large numbers* and the assertion that $\bar{X}_n \longrightarrow E[X_1]$ in probability is called the *weak law of large numbers*. See Karandikar [4] and Feller [5] for extensive discussions (including proofs) on both the weak law and the strong law of large numbers.

3. Typical Sets and the Asymptotic Equipartition Theorem

As a consequence of the law of large numbers, we have the following result:

Theorem 3.1. If $X_1, X_2, \ldots$ are independent and identically distributed (i.i.d.) random variables with a probability distribution $p(.)$, then

$$-\frac{1}{n} \log p(X_1, X_2, \ldots, X_n) \;\to\; H(X)$$

with probability 1.

Proof. Since $X_1, \ldots, X_n$ are i.i.d. random variables,

$p(X_1, \ldots, X_n) = \prod_i p(X_i)$. Hence,

$$-\frac{1}{n}\log p(X_1, X_2, \ldots, X_n) = -\frac{1}{n}\sum_i \log p(X_i)$$

converges to $-E[\log p(X)] = H(X)$ with probability 1 as $n \longrightarrow \infty$, by the strong law of large numbers. ■

This theorem says that the quantity $\frac{1}{n}\log\frac{1}{p(X_1,X_2\ldots,X_n)}$ is close to the entropy $H(X)$ when n is large. Therefore, it makes sense to divide the set of sequences $(x_1, x_2, \ldots, x_n) \in \mathcal{X}^n$ into two sets: the *typical set* wherein each sequence $(x_1, x_2, \ldots, x_n)$ is such that $\frac{1}{n}\log\frac{1}{p(x_1,\ldots,x_n)}$ is close to the entropy $H(X)$ and the non-typical set containing sequences with $\frac{1}{n}\log\frac{1}{p(x_1,\ldots,x_n)}$ bounded away from the entropy $H(X)$. As n increases, the probability of sequences in the typical set becomes high.

Definition 3.1. For each $\epsilon > 0$, let $A_\epsilon^{(n)}$ be the set of sequences $(x_1, x_2, \ldots, x_n) \in \mathcal{X}^n$ with the following property:

$$2^{-n(H(X)+\epsilon)} \leq p(x_1, x_2, \ldots, x_n) \leq 2^{-n(H(x)-\epsilon)}.$$

Call $A_\epsilon^{(n)}$ a typical set.

Theorem 3.2. Fix $\epsilon > 0$. For $A_\epsilon^{(n)}$ as defined above the following hold:

1. If $(x_1, x_2, \ldots, x_n) \in A_\epsilon^{(n)}$, then $H(X) - \epsilon \leq -\frac{1}{n}\log p(x_1, x_2, \ldots, x_n) \leq H(X) + \epsilon$.
2. If $\{X_1, X_2, \ldots X_n\}$ are i.i.d. with probability distribution $p(X)$ then $Pr((x_1, x_2, \ldots x_n) \in A_\epsilon^{(n)}) \equiv Pr(A_\epsilon^{(n)}) \longrightarrow 1$ as $n \longrightarrow \infty$.
3. $|A_\epsilon^{(n)}| \leq 2^{n(H(X)+\epsilon)}$, where for any set S, $|S|$ is the number of elements in S.
4. $|A_\epsilon^{(n)}| \geq (1-\epsilon)2^{n(H(X)-\epsilon)}$ for n sufficiently large.

Shannon's source-coding theorem states that to describe a source by a code C, on an average at least $H(X)$ bits (or, D-ary symbols) are required.

Proof. The proof of (1) follows from the definition of $A_\epsilon^{(n)}$ and that of (2) from Theorem 3.1.

To prove (3), we observe that the following set of equations hold:

$$1 = \sum_{(x_1,\ldots,x_n)\in\mathcal{X}^n} p(x_1,\ldots,x_n) \geq \sum_{(x_1,\ldots,x_n)\in A_\epsilon^{(n)}} p(x_1,\ldots,x_n)$$

$$\geq \sum_{(x_1,\ldots,x_n)\in A_\epsilon^{(n)}} 2^{-n(H(X)+\epsilon)} = 2^{-n(H(X)+\epsilon)}|A_\epsilon^{(n)}|.$$

To prove (4), note by (2) that for n large enough

$$Pr((x_1,\ldots,x_n) \in A_\epsilon^{(n)}) > 1-\epsilon.$$

This means that

$$1-\epsilon < Pr((x_1,\ldots,x_n) \in A_\epsilon^{(n)}) =$$

$$\sum_{(x_1,\ldots,x_n)\in A_\epsilon^{(n)}} p(x_1,x_2,\ldots x_n)$$

$$\leq 2^{-n(H(X)-\epsilon)}|A_\epsilon^{(n)}|.$$

■

4. Source Coding Theorem

Shannon's source-coding theorem states that to describe a source by a code C, on an average at least $H(X)$ bits (or, D-ary symbols) are required. The theorem further states that there exist codes that have an expected code-length very close to $H(X)$. The following theorem shows the existence of codes with expected code-length close to the source entropy. This theorem can also be called the *achievability*-part of Shannon's source-coding theorem, and says that the average number of bits needed to represent a symbol x from the source is $H(X)$, the entropy of the source.

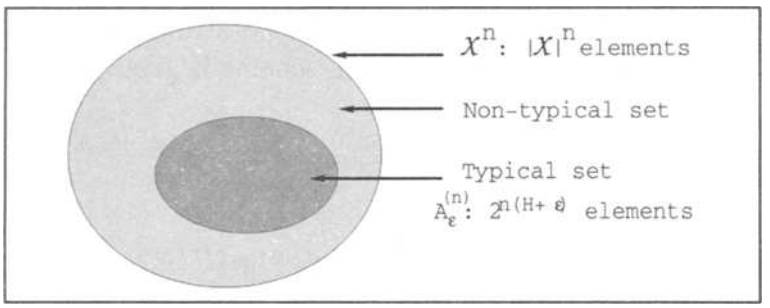

Figure 1. Typical sets and source-coding.

Theorem 4.1. Let $X^n = (X_1, X_2, \ldots X_n)$ be an independent and identically distributed (i.i.d.) source with distribution $p(X)$. Let $\eta > 0$. Then there exists an $n_0 = n_0(\eta)$, depending on η such that for all $n \geq n_0$, there exists a code which maps sequences $x^n = (x_1, x_2, \ldots, x_n)$ of length n into binary strings such that the mapping is one-to-one (and therefore invertible) and

$$E[\frac{1}{n}\ell(X^n)] \leq H(X) + \eta,$$

where $\ell(x^n)$ is the length of the code representing x^n.

Proof. Let $X_1, X_2, \ldots, X_n$ be i.i.d. random variables chosen from the probability distribution $p(X)$. The goal is to find short descriptions for such sequences of random variables. Consider the following encoding scheme: For a chosen $\epsilon > 0$ and integer $n > 0$ (both to be chosen later) partition the sequences in $\mathcal{X}^n$ into two sets: the typical set $A_\epsilon^{(n)}$ and its complement as shown in *Figure* 1. Order all the elements in each set according to some order. Then we can represent each sequence of $A_\epsilon^{(n)}$ by giving the index of that sequence in the set. From Theorem 3.2, there are at most $2^{n(H(X)+\epsilon)}$ such sequences, and hence, the indexing of elements in $A_\epsilon^{(n)}$ requires no more than $\lceil n(H(X)+\epsilon)\rceil \leq n(H(X)+\epsilon)+1$ bits. Similarly, the sequences not in $A_\epsilon^{(n)}$ can be indexed by $\lceil n \log|\mathcal{X}|\rceil$ $\leq n \log|\mathcal{X}|+1$ bits since the number of sequences not in the typical set is less than $|\mathcal{X}|^n$. Suppose we prefix every code sequence in $A_\epsilon^{(n)}$ by a 0 and every code sequence not in $A_\epsilon^{(n)}$ by a 1, then we can uniquely represent every

Let $X_1, X_2, \ldots, X_n$ be i.i.d. random variables chosen from the probability distribution $p(X)$. The goal is to find short descriptions for such sequences of random variables.

The converse to the source-coding theorem states that every code that maps sequences of length n from a source X into binary strings, such that the mapping is one-to-one, has an average code length that is at least the entropy $H(X)$.

sequence of length n from $\mathcal{X}^n$. The average length of codewords in such an encoding scheme is given by

$$E[\ell(X^n)] = \sum_{x^n \in \mathcal{X}^n} p(x^n)\ell(x^n) = \sum_{x^n \in A_\epsilon^{(n)}} p(x^n)\ell(x^n) +$$

$$\sum_{x^n \in \mathcal{X}^n - A_\epsilon^{(n)}} p(x^n)\ell(x^n) \leq \sum_{x^n \in A_\epsilon^{(n)}} p(x^n)[n(H(X)+\epsilon)+2] +$$

$$\sum_{x^n \in \mathcal{X}^n - A_\epsilon^{(n)}} p(x^n)[n \log |\mathcal{X}| + 2]$$

$$= Pr\{A_\epsilon^{(n)}\}[n(H(X)+\epsilon)+2] + (1 - Pr\{A_\epsilon^{(n)}\})[n \log |\mathcal{X}|+2]$$

$$= Pr\{A_\epsilon^{(n)}\}[n(H(X)+\epsilon)] + (1 - Pr\{A_\epsilon^{(n)}\})[n \log |\mathcal{X}|] + 2.$$

By (2) of Theorem 3.2, there exists a $n_0(\epsilon)$ such that for $n \geq n_0(\epsilon), 1 - Pr(A_\epsilon^{(n)}) < \epsilon$. Thus

$$E[l(X^n)] \leq n(H(X)+\epsilon) + n\epsilon(\log |\mathcal{X}|) + 2 = n(H(X)+\epsilon').$$

where $\epsilon' \equiv \epsilon + \epsilon \log |\mathcal{X}| + \frac{2}{n}$. Given $\eta > 0$ choose an ϵ and $n_0(\epsilon)$ large enough such that $\epsilon' < \eta$. Since the choice of ϵ depends on η, $n_0(\epsilon)$ can be written as $n_0(\eta)$. ■

The above result proves the *existence* of a source-coding scheme which achieves an average code length of $(H(X)+\epsilon')$. However, this proof is non-constructive since it does not describe how to explicitly find such a typical set. The converse to the source-coding theorem states that every code that maps sequences of length n from a source X into binary strings, such that the mapping is one-to-one, has an average code length that is at least the entropy $H(X)$. We will prove another version of this result in the following section by restricting the coding to a special class of codes known as prefix-free codes.

5. Efficient Coding of Information

The notion of trees is introduced to illustrate how a code, that maps a sequence of symbols from a source X

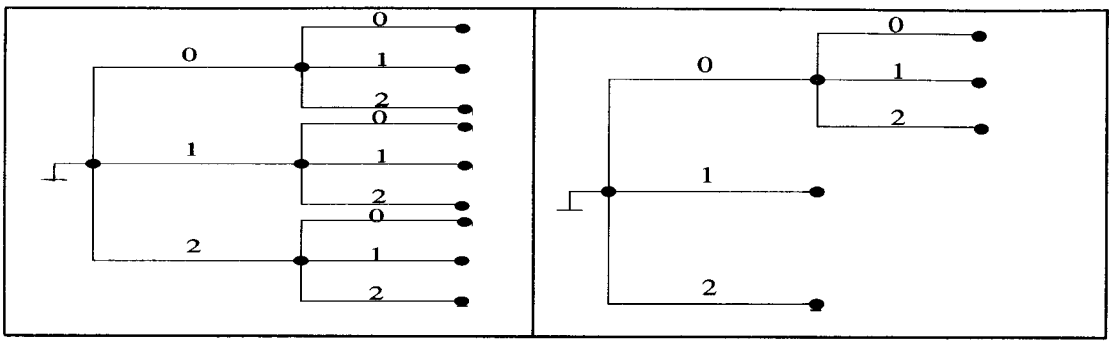

Figure 2. A complete 3-ary tree of length 2.

Figure 3. An incomplete 3-ary tree of length 2.

onto binary, or more generally, D-ary strings, may be described. This notion will further enable us to provide coding techniques that are described by building appropriate trees and labeling the edges of the tree so that the codewords of the code are specified by taking sequences of edge-labels on this tree.

Definition 5.1. A D-ary tree is a finite (or semi-infinite) rooted tree such that D branches stem outward from each node.

Definition 5.2. The complete D-ary tree of length N is the D-ary tree with D^N leaves, each at depth N from the root node.

Figures 2 and 3 respectively, show a complete 3-ary tree and an incomplete 3-ary tree of length $N = 2$.

Definition 5.3. A code is *prefix-free* if no codeword in the code is a prefix of another codeword in the same code.

As an example, the code with codewords 0 and 011 is not prefix-free. A prefix-free code has the property that a codeword is recognizable as soon as its last digit is known. This ensures that if a sequence of digits corresponding to several codewords in the code are written contiguously, then the codewords from the sequence can be read out instantaneously. For example, suppose the codewords of a code are $u_1 = 0$, $u_2 = 10$, $u_3 = 110$ and $u_4 = 111$. Then the following sequence of digits 011010010 corresponds to the codewords u_1, u_3, u_2, u_1, u_2.

A prefix-free code has the property that a codeword is recognizable as soon as its last digit is known.

There exists a D-ary prefix-free code whose code-lengths are the positive integers $w_1, w_2, \ldots, w_K$ if and only if $\sum_{i=1}^{K} D^{-w_i} \leq 1$.

For the rest of the article, we restrict our attention to prefix-free codes. Every D-ary prefix-free code can be represented on a D-ary tree wherein the set of codewords in the code correspond to a set of leaf nodes in the D-ary tree and the length of a codeword is equal to the depth of the corresponding leaf node from the root node. We now state a necessary and sufficient condition for the existence of prefix-free codes.

Lemma 5.1 *(Kraft's Inequality):* There exists a D-ary prefix-free code whose code-lengths are the positive integers $w_1, w_2, \ldots, w_K$ if and only if

$$\sum_{i=1}^{K} D^{-w_i} \leq 1.$$

Proof. Observe that in a complete D-ary tree of length N, D^{N-w} leaves stem from each node that is at depth w from the root node, for $w < N$.

Suppose there exists a D-ary prefix-free code whose code-lengths are $w_1, w_2, \ldots, w_K$. Without loss of generality, let $w_1 \leq w_2 \leq \ldots \leq w_K$. Let $N = \max_i w_i = w_K$. We will now construct a tree for the prefix-free code starting with a complete D-ary tree of length N. Order the leaves at depth N in some fashion. Starting with $i = 1$ and the first leaf node z_1, if $w_1 < N$, identify the node, say v, in the tree that is at depth w_1 and from where the leaf node z_1 stems. Delete the portion of the tree from v down to its leaves. By this process, we make this vertex v a leaf node at depth w_1. For $i = 2$, identify the next leaf node z_2 not equal to v. Find a node v_2 at depth w_2 from which z_2 stems and delete the subtree from v_2 down to its leaves, thereby making v_2 a leaf node. Similarly, repeat the process for $i = 3, 4, \ldots, K$. By this process, at each step, D^{N-w_i} leaf nodes in the original complete D-ary tree are deleted. However, since the complete D-ary tree contained only D^N leaf nodes, we have $D^{N-w_1} + D^{N-w_2} + \ldots + D^{N-w_K} \leq D^N$. Dividing by D^N gives the desired condition.

To prove the converse, suppose $w_1, w_2, \ldots, w_K$ are positive integers satisfying the inequality $\sum_{i=1}^{K} D^{-w_i} \leq 1$ $(*)$. Then, without loss of generality, let $w_1 \leq w_2 \leq \ldots \leq w_K$. Set $N = \max_i w_i = w_K$. We want to show that we are able to construct a prefix-free code with codeword lengths $w_1, w_2, \ldots, w_K$. Start again with the complete D-ary tree.

Let $i = 1$. Identify a node z_1 at depth w_1 to be the first codeword of the code we intend to construct. If $w_1 < N$, delete the subtree stemming from z_1 down to its leaves so that z_1 becomes a leaf node. Next, increment i by 1 and look for a node z_i in the remaining tree which is not a codeword and is at depth w_i. If there is such a node, call this the ith codeword and delete the subtree stemming from z_i down to its leaves. Repeat the above process by incrementing i by 1. This algorithm terminates when either $i > K$ or when the algorithm cannot find a node z_i satisfying the desired property in the ith step for $i \leq K$.

If the algorithm identifies K codeword nodes, then a prefix-free code can be constructed by labeling the edges of the tree as follows: At each node, label the $r \leq D$ edges that stem from it as $0, 1, 2 \ldots, r-1$ in any order. The ith codeword is then the sequence of edge-labels from the root node up to the ith codeword node.

The only part remaining to be seen is that the above algorithm will not terminate before K codeword nodes have been identified. To show this, consider the ith step of this algorithm when $z_1, z_2, \ldots, z_{i-1}$ have been identified and z_i needs to be chosen. The number of surviving leaves at depth N not stemming from any codeword is $D^N - (D^{N-w_1} + D^{N-w_2} + \ldots + D^{N-w_{i-1}})$. But by our assumption $(*)$, this number is greater than zero. Thus, there exists a node z_i which is not a codeword and which is at depth w_i in the tree that remains at step i. Furthermore, since $w_i \geq w_{i-1} \geq w_{i-2} \ldots \geq w_1$, the node z_i will not lie in any of the paths from the root node to any of

If the algorithm identifies K codeword nodes, then a prefix-free code can be constructed by labeling the edges of the tree as follows: At each node, label the $r \leq D$ edges that stem from it as 0,1,2 ..., r–1 in any order. The i th codeword is then the sequence of edge-labels from the root node up to the i th codeword node.

The depth of a codeword-node is equal to the length of the codeword in the code, and the codeword is the sequence of edge-labels on this tree from the root node up to the corresponding codeword-node.

$z_1, z_2 \ldots, z_{i-1}$. This argument holds for $i = 1, 2, \ldots, K$; hence, this algorithm will always be able to identify K codeword nodes and construct the corresponding prefix-free code. ■

A. *Trees with Probability Assignments*

Let us suppose that there is a D-ary prefix-free code that maps symbols from a source U onto D-ary strings such that the mapping is invertible. For convenience, let $u_1, u_2, \ldots, u_K$ be the source symbols and let the codewords corresponding to these symbols also be denoted by $u_1, u_2, \ldots, u_K$. The proof of the Kraft inequality tells us how to specify a D-ary prefix-free code on a D-ary tree. That is, a D-ary tree can be built with the edges labeled as in the proof of Lemma 5.1 such that the codewords are specified by certain leaf nodes on this tree, the depth of a codeword-node is equal to the length of the codeword in the code, and the codeword is the sequence of edge-labels on this tree from the root node up to the corresponding codeword-node. Furthermore, probabilities can be assigned to all the nodes in the tree as follows: assign the probability $p(u_i)$ for the codeword node which corresponds to the codeword u_i (or, the source symbol u_i.) Assign the probability zero to a leaf node that is not a codeword node. Assign a parent node (that is not a leaf node), the sum of the probabilities of its children nodes. It is easy to verify that for such an assignment, the root node will always have a probability equal to 1. Such a D-ary tree gives a complete description of the D-ary prefix-free code. This description will be useful in describing two specific coding methods: the Shannon-Fano coding algorithm and the Huffman coding algorithm.

B. *Source Coding Theorem for Prefix-Free Codes*

Before we present the two coding techniques, we present another version of the source-coding theorem for the

special case of prefix-free codes:

Theorem 5.1. Let $\ell_1^*, \ell_2^*, \ldots, \ell_m^*$ be the optimal codeword lengths for a source with distribution $p(X)$ and a D-ary alphabet, and let L^* be the corresponding expected length of the optimal prefix-free code ($L^* = \sum_i p_i \ell_i^*$). Then

$$H_D(X) \leq L^* \leq H_D(X) + 1.$$

The expected length L of any prefix-free D-ary code for a source described by the random variable X with distribution $p(X)$ is at least the entropy $H_D(X)$.

Proof. We first show that the expected length L of any prefix-free D-ary code for a source described by the random variable X with distribution $p(X)$ is at least the entropy $H_D(X)$. To see this, let the code have codewords of length $\ell_1, \ell_2, \ldots$ and let the corresponding probabilities of the codewords be $p_1, p_2, \ldots$. Consider the difference between the expected length L and the source entropy $H_D(X)$,

$$L - H_D(X) = \sum_i p_i \ell_i - \sum_i p_i \log_D \frac{1}{p_i}$$

$$= -\sum_i p_i \log_D D^{-\ell_i} + \sum_i p_i \log_D p_i.$$

Let $r_i = D^{-\ell_i}/(\sum_j D^{-\ell_j})$ and $c = \sum_j D^{-\ell_j}$. Then,

$$L - H_D(X) = \sum_i p_i \log_D \frac{p_i}{r_i} - \log_D c.$$

The first term on the right-hand side is a well-known term in information theory, called the 'divergence' between the probability distributions r and p can be shown to be non-negative. (See [2]). The second term is also non-negative since $c \leq 1$ by the Kraft inequality. This shows that $L - H_D(X) \geq 0$.

Suppose there is a prefix-free code with codeword lengths $\ell_1, \ell_2, \ldots$ such that $\ell_i = \lceil \log_D \frac{1}{p_i} \rceil$. Then the ℓ_i's satisfy the Kraft inequality since $\sum_i D^{-\ell_i} \leq \sum_i p_i = 1$. Furthermore, we have

$$\log_D \frac{1}{p_i} \leq \ell_i \leq \log_D \frac{1}{p_i} + 1.$$

This theorem says that an optimal code will use at most one bit more than the source entropy to describe the source alphabets.

Hence, the expected length L of such a code satisfies

$$H_D(X) = \sum_i p_i \log_D \frac{1}{p_i} \leq L =$$

$$\sum_i p_i \ell_i \leq \sum_i p_i (\log_D \frac{1}{p_i} + 1) = H_D(X) + 1.$$

Since an optimal prefix-free code can only be better than the above code, the expected length L^* of an optimal code satisfies the inequality $L^* \leq L \leq H_D(X)+1$. Since we have already shown that any prefix-free code must have an expected length which is at least the entropy, this proves that

$$H_D(X) \leq L^* \leq H_D(X) + 1. \quad \blacksquare$$

Note that in the above theorem, we are trying to find a description of all the alphabets of the source individually. This theorem says that an optimal code will use at most one bit more than the source entropy to describe the source alphabets. In order to achieve a better compression, we can find a corresponding code for a new source whose alphabets are sequences of length $n > 1$ of the original source X. As n gets larger, the corresponding optimal code will have an effective code-length as close to its source entropy as possible. In fact, it can be shown that the effective code-length for an optimal code then satisfies $H(X) \leq E[\frac{1}{n}X^n]^* \leq H(X) + \frac{1}{n}$. This result is in accordance with Theorem 4.1 of Section 4.

C. *Shannon–Fano Prefix-Free Codes*

Following the proof-technique of Theorem 5.1, we present a natural source-coding algorithm that is popularly known as the Shannon–Fano algorithm. For a source U, this algorithm constructs D-ary prefix-free codes with expected length at most one more than the entropy of the source. The algorithm is as follows:

(*i*) Suppose the source U is described by the source symbols $u_1, u_2, \ldots, u_K$ having probability of occurrences $p_1 = p(u_1), p_2 = p(u_2), \ldots, p_K = p(u_K)$, respectively. Then, for $i = 1, 2, \ldots, K$, set $w_i = \lceil \log_D \frac{1}{p_i} \rceil$ to be the length of the codeword corresponding to u_i. Observe that the sequence of w_i's satisfies the Kraft inequality since

$$\sum_i D^{-w_i} \leq \sum_i p_i = 1.$$

The effective code length of the Shannon–Fano code is then $\Sigma_i p_i w_i$. This quantity is less than $H_D(U)+1$ from the proof of Theorem 5.1.

(*ii*) Using the algorithm mentioned in the proof of the Kraft-inequality (Lemma 5.1), a D-ary tree is grown such that the codeword nodes are at depths $w_1, w_2, \ldots, w_K$ on this tree. The sequence of edge labels from the root node to a codeword node specify the codeword of the Shannon–Fano code.

We can further assign probabilities to the nodes of this tree as mentioned before. The effective code length of the Shannon–Fano code is then $\sum_i p_i w_i$. This quantity is less than $H_D(U) + 1$ from the proof of Theorem 5.1.

Example 5.1. Consider a source U with alphabet set $\mathcal{U} = \{u_1, u_2, u_3, u_4, u_5\}$ with the probability of occurrences $p(u_1) = 0.05, p(u_2) = 0.1, p(u_3) = 0.2, p(u_4) = 0.32, p(u_5) = 0.33$. For $D = 2$, the lengths of the codewords for a Shannon–Fano binary prefix free code are $w_1 = \lceil \log_2 \frac{1}{0.05} \rceil = 5$, $w_2 = \lceil \log_2 \frac{1}{0.1} \rceil = 4$, $w_3 = \lceil \log_2 \frac{1}{0.2} \rceil = 3$, $w_4 = \lceil \log_2 \frac{1}{0.32} \rceil = 2$, and $w_5 = \lceil \log_2 \frac{1}{0.33} \rceil = 2$. The Shannon–Fano algorithm constructs a ($D = 2$) binary tree as shown in *Figure* 4. The codewords corresponding to u_1, u_2, u_3, u_4, u_5 are 00000, 0001,010, 10, 11, respectively. The average code-length is $5 \cdot 0.05 + 4 \cdot 0.1 + 3 \cdot 0.2 + 2 \cdot 0.32 + 2 \cdot 0.33 = 2.55$ bits and the entropy of the source is 2.0665 bits.

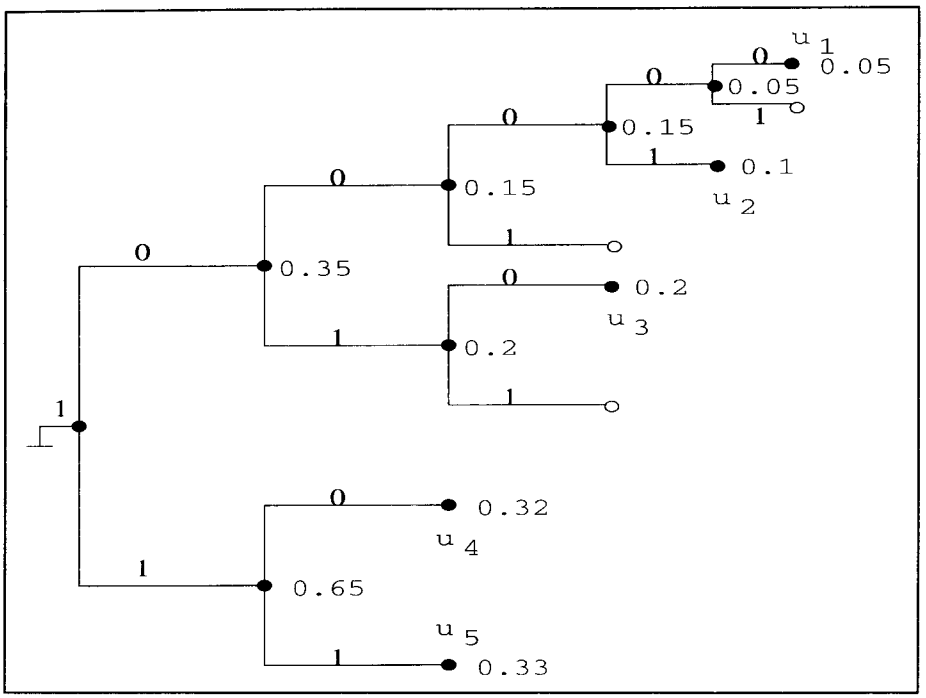

Figure 4. A Shannon–Fano binary prefix-free code.

6. Huffman Coding

This section presents Huffman's algorithm for constructing a prefix-free code for a source U. The binary case is considered first where we obtain an optimal binary prefix-free code. The more general D-ary case, for $D > 2$, is considered next.

A. *Binary Case*

Using the notion of trees with probabilities, the objective of Huffman coding is to construct a binary tree with probability assignments such that a chosen set of leaves in this tree are codewords. We will first show that to obtain an optimal code, the binary tree that we construct must satisfy the following two lemmas.

The objective of Huffman coding is to construct a binary tree with probability assignments such that a chosen set of leaves in this tree are codewords.

Lemma 6.1. The binary tree of an optimal binary prefix-free code for U has no unused leaves.

Proof. Suppose to the contrary that there is a leaf node v in the binary tree that is not a codeword, then there

is a parent node v' that has v as its child node. Since the tree is binary, v has another child node u. Suppose u is a codeword node corresponding to the codeword c, then deleting u and v and representing v' as the codeword node c yields a code with a lower average code-length since we have reduced the code-length for the codeword c without affecting the code lengths of the remaining codewords. However, since we assumed that the binary tree represented an optimal code to begin with, this is a contradiction. ■

Lemma 6.2. There is an optimal binary prefix-free code for U such that the two least likely codewords, say u_{K-1} and u_K, differ only in their last digit.

Proof. Suppose we have the binary tree for an optimal code. Let u_i be one of the longest codewords in the tree. By Lemma 6.1, there are no unused leaves in this tree, so there is another codeword u_j which also has the same parent node as u_i. Suppose $u_j \neq u_K$, then we can interchange the codeword nodes for u_j and u_K. This interchange will not increase the average code-length since $p(u_K) \leq p(u_j)$ and since the code-length of u_K is at least the code-length of u_j in the original code. Now if $u_i \neq u_{K-1}$, we can similarly interchange the codeword nodes for u_i and u_{K-1}. Thus, the new code has u_K and u_{K-1} among its longest codewords and u_K and u_{K-1} are connected to the same parent node. Since the codewords are obtained by the sequence of edge-labels from the root node up to the corresponding codeword nodes, the codewords for u_K and u_{K-1} differ only in their last digit. ■

The above two lemmas suggest that to construct an optimal binary tree, it is useful to begin with the two least likely codewords. Assigning these two codewords as leaf nodes that are connected to a common parent node gives part of the tree for the optimal code. The parent node is now considered as a codeword having a probability that

The above two lemmas suggest that to construct an optimal binary tree, it is useful to begin with the two least likely codewords.

Consider the problem of twenty questions where we wish to find an efficient series of yes-or-no questions to determine an object from a class of objects. Suppose the probability distribution of the objects in this class is known *a priori* and it is assumed that any question depends on the series of answers received before that question is to be posed, then the Huffman coding algorithm provides an optimal solution to this problem.

is the sum of the probabilities of the two leaf nodes and the two codewords corresponding to the leaf nodes are now ignored. The algorithm proceeds as before by considering the two least likely codewords that are available in the new code and constructs the subtree corresponding to these vertices as before.

Huffman's algorithm for constructing an optimal binary prefix-free code for a source U with K symbols is summarized below. It is assumed that $P(u) \neq 0$ for all $u \in \mathcal{U}$. That is, no codeword will be assigned to a symbol $u \in \mathcal{U}$ that has a probability of occurrence equal to zero.

(i) Assign labels, i.e., $u_1, u_2, \ldots, u_K$ to K vertices which will be the leaves of the final tree, and assign the probability $P(u_i)$ to the vertex labeled u_i, for $i = 1, \ldots, K$. Let these K vertices be called 'active' vertices.

(ii) Create a new node v and join the two least likely active vertices u_i and u_j (i.e., two vertices with the least probabilities) to v. That is connect v to u_i and u_j. Assign the labels 0 and 1 to the two edges (v, u_i) and (v, u_j) in any order.

(iii) Deactivate the two vertices connected to v and activate the node v. Assign the new active vertex v the sum of the probabilities of u_i and u_j.

(iv) If there is only one active vertex, then call this the root vertex and stop. Otherwise, return to Step (ii).

Example 6.1. Consider a source U with alphabet set $\mathcal{U} = \{u_1, u_2, u_3, u_4, u_5\}$ with the probability of occurrences $p(u_1) = 0.05$, $p(u_2) = 0.1, p(u_3) = 0.2, p(u_4) = 0.32, p(u_5) = 0.33$. The Huffman coding algorithm constructs a binary tree as shown in *Figure* 5. The codewords corresponding to u_1, u_2, u_3, u_4, u_5 are 000, 001,01,

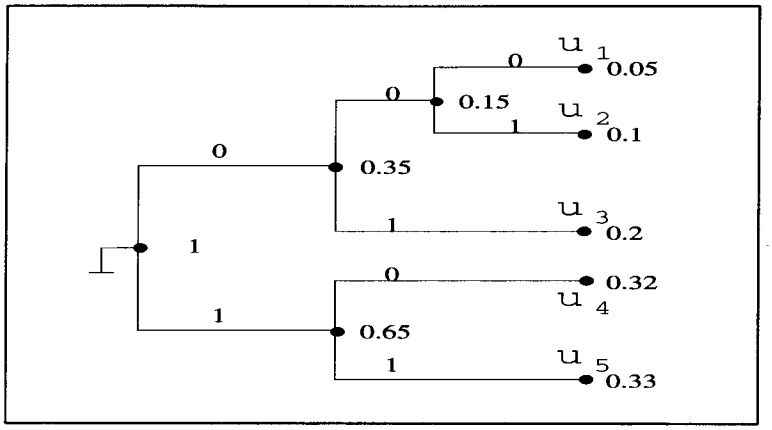

Figure 5. An (optimal) binary Huffman code.

10, 11, respectively. The average code-length is $3 \cdot 0.05 + 3 \cdot 0.1 + 2 \cdot 0.2 + 2 \cdot 0.32 + 2 \cdot 0.33 = 2.15$ bits. Note that this number is smaller than the average code-length of the Shannon–Fano code in Example 5.1.

Example 6.2. Consider the problem of twenty questions where we wish to find an efficient series of yes-or-no questions to determine an object from a class of objects. Suppose the probability distribution of the objects in this class is known *a priori* and it is assumed that any question depends on the series of answers received before that question is to be posed, then the Huffman coding algorithm provides an optimal solution to this problem.

B. *D-ary case*

The algorithm for the binary case can be easily generalized to construct an optimum D-ary prefix-free code. The following two lemmas, which are the extensions of Lemmas 6.1 and 6.2 for the D-ary case, are stated without proof [3].

Lemma 6.3. The number of unused leaves for an optimal D-ary prefix-free code for a source U with K possible values, $K \geq D$, is the remainder when $(K - D)(D - 2)$ is divided by $D - 1$.

Lemma 6.4. There is an optimal D-ary prefix-free code for a source U with K possible values such that the $D-r$

There is an optimal D-ary prefix-free code for a source U with K possible values such that the D–r least likely codewords differ only in their last digit, where r is the remainder when $(K–D)(D–2)$ is divided by D–1.

least likely codewords differ only in their last digit, where r is the remainder when $(K-D)(D-2)$ is divided by $D-1$.

Based on the two lemmas, Huffman's algorithm for constructing an optimal D-ary prefix-free code ($D \geq 3$) for a source U with K symbols, such that $P(u) \neq 0$ for all u, is given by:

(i) Label K vertices as $u_1, u_2, \ldots, u_K$ and assign the probability $P(u_i)$ to the vertex labeled u_i, for $i = 1, 2, \ldots, K$. Call these K vertices 'active'. Let r be the remainder when $(K-D)(D-2)$ is divided by $D-1$.

(ii) Form a new node v and connect the $D-r$ least likely active vertices to v via $D-r$ edges. Label these edges as $0, 1, 2, \ldots, D-r-1$ in any random order.

(iii) Deactivate the $D-r$ active vertices that are connected to the new node v and activate v. Assign v the sum of the probabilities of the $D-r$ deactivated vertices.

(iv) If there is only one active vertex remaining, then call this vertex the root node and stop. Otherwise, return to Step (ii).

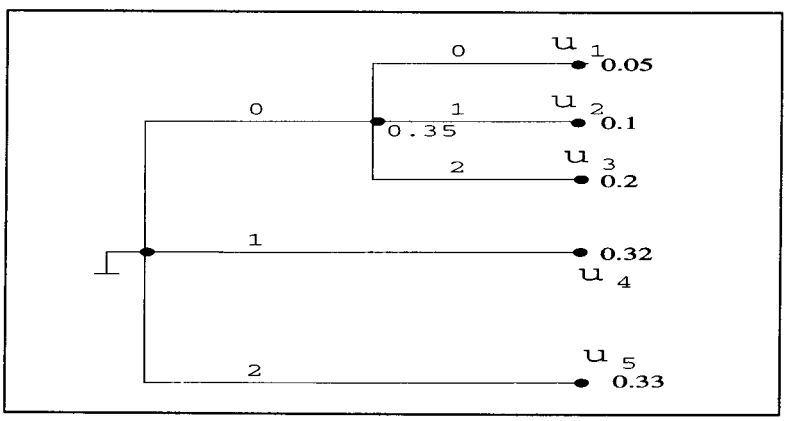

Figure 6. An (optimal) 3-ary Huffman code.

Example 6.3. Consider the same source U as in Example 6.1. The Huffman coding algorithm constructs a 3-ary prefix-free code as shown in *Figure* 6. The codewords corresponding to u_1, u_2, u_3, u_4, u_5 are 00, 01,02, 1, 2, respectively. The average code-length is $2 \cdot 0.05 + 2 \cdot 0.1 + 2 \cdot 0.2 + 1 \cdot 0.32 + 1 \cdot 0.33 = 1.35$ ternary digits and the entropy of the source is $H_3(U) = 1.3038$ ternary digits.

Remark 6.1. While the proof of Theorem 5.1 seems to suggest that the Shannon–Fano coding technique is a very natural technique to construct a prefix-free code, it turns out that this technique is not optimal. To illustrate the sub-optimality of this technique, consider a source with two symbols u_1 and u_2 with probability of occurrences $p(u_1) = 1/16$ and $p(u_2) = 15/16$. The Shannon–Fano algorithm finds a binary prefix-free code with codeword lengths equal to $\lceil \log_2 \frac{1}{1/16} \rceil = 4$ and $\lceil \log_2 \frac{1}{15/16} \rceil = 1$, respectively, whereas the Huffman-coding algorithm yields a binary prefix-free code with codeword lengths equal to 1 for both.

7. Summary

This article has presented a derivation of the best achievable performance for a source encoder in terms of the average code length. The source-coding theorem for the case of prefix-free codes shows that the best coding scheme for a source X has an expected code-length bounded between $H(X)$ and $H(X) + 1$. Two coding techniques, the Shannon–Fano technique and the Huffman technique, yield prefix-free codes which have expected code-lengths at most one more than the source entropy. The Huffman coding technique yields an optimal prefix-free code yielding a lower expected code-length compared to a corresponding Shannon–Fano code. However, the complete proof of the optimality of Huffman coding [2] is not presented here; Lemma 6.1 only presents a necessary condition that an optimal code must satisfy.

Suggested Reading

[1] C E Shannon, A Mathematical Theory of Communications, *Bell System Technical Journal*, Vol. 27, pp.379-423, July, 1948; Vol.27, pp. 623-656, October 1948.

[2] T M Cover and J A Thomas, *Elements of Information Theory*, Wiley Series in Telecommunications, John Wiley & Sons, Inc., New York, 1991.

[3] J L Massey, *Applied Digital Information Theory I.* Lecture Notes, ETH, Zurich. Available online at http://www.isi.ee.ethz.ch/education/public/pdfs/aditI.pdf

[4] R L Karandikar, On Randomness and Probability: How to Model Uncertain Events Mathematically, *Resonance*, Vol.1, No.2, Feb. 1996.

[5] W Feller, *An Introduction to Probability Theory and its Applications*, Vols.1,2, Wiley-Eastern, 1991.

[6] S Natarajan, Entropy, Coding and Data Compression, *Resonance*, Vol.6, No.9, 2001.

Address for Correspondence
Deepak Sridhara
Institut für Mathematik
Universität Zürich
Email:
sridhara@math.unizh.ch